Engineering High Assurance Distributed Cyber Physical Systems

Scott A. Hissam, David S. Kyle, Sagar Chaki, Dionisio de Niz, Jeffery P. Hansen, Gabriel Moreno, and Mark Klein
Carnegie Mellon University, Pittsburgh, PA, USA
{shissam,dskyle,chaki,dionisio,jhansen,gmoreno,mk}@sei.cmu.edu

***Abstract*— Distributed Adaptive Real-Time (DART) systems are interconnected and collaborating systems that continuously must satisfy guaranteed and highly critical requirements (e.g., collision avoidance), while at the same time adapting, smartly, to achieve best-effort and low-critical application requirements (e.g., protection coverage) when operating in dynamic and uncertain environments. This paper introduces our approach to engineering a DART system so that we achieve high assurance in its runtime behavior against a set of formally specified requirements. It describes our technique to: (i) ensure asymmetric timing protection between high- and low-critical threads (HCTs and LCTs) on each node in the DART system, and (ii) verify that the self-adaptation within, and coordination between, the nodes and their interaction with the physical environment do not violate high and low requirements. We present our ongoing research to integrate advances in model-based engineering with compositional analysis techniques to formally verify safety-critical properties demanded in safety-conscience domains such as aviation and automotive, and introduce our DART model problem that serves as an end-to-end demonstration of our integrated engineering approach.**

I. INTRODUCTION

Development, testing, and operation of any software system in a correct, cost-effective, and timely manner is the essence of software engineering [1]. To simply appreciate the complexities and challenges in addressing the cost and timeliness of the engineering processes, one need only look to the community of research and practice embodied by conferences dedicated to such topics, including, but not only, International Conference on Software Engineering, Fundamental Approaches to Software Engineering, and Foundations of Software Engineering. However, ensuring that the software is correct, or *verifiably correct*, is arguably more challenging and complex. Such a statement is anecdotally supported by the notion that software development and delivery is often delayed (at an additional cost) to ensure that the software is correct—be it at requirements elicitation, architecture and design development, implementation and review steps, or test, integration and verification activities. Furthermore, the sheer number of conferences (both large and small) that are dedicated to the subfields of software engineering that focus on the correctness and verification of correctness is well beyond proper elaboration here [2].

Disastrous failures in embedded systems which interact with the physical world have demonstrated the consequences of not adequately verifying the correctness of the software (such as Therac-25, Swedish JAS 39 Gripen, Boeing V-22 Osprey, and Airbus A320-200). Embedded systems with critical runtime properties are becoming increasingly distributed, consisting of interconnected nodes (i.e., multi-agent) that collaboratively provide more capability. They use self-adaptation to achieve their goals when operating in uncertain environments. However, coordination, adaptation, and uncertainty pose key challenges for assuring the safety and application critical behavior of such distributed adaptive real-time (DART) systems. These challenges are exemplified by:

- Timeliness: performing the right function, and doing so at the *right* time.
- Resource Constraints: limits with respect to power, weight, bandwidth, connectivity, storage, and calculations per second.
- Sensor rich: sensing the physical world with sensors that have varying fidelity and can fail.
- Intimate cyber-physical interactions: the physical world is continuous and the digital interface cannot account for everything.
- Autonomous behavior: adapting smartly to events within an agent (e.g., failure), between agents (e.g., loss of a peer) and external (e.g., unforeseeable).
- Computationally complex decisions: number of interacting agents and co-dependent decisions made in real-time without causing interference.

To engineer a high assurance DART system for safety-conscience systems, we present our ongoing research to integrate compositional analysis techniques with model-based engineering to address these challenges and introduce our DART model problem that serves as an end-to-end demonstration of our integrated engineering approach.

II. RELATED WORK

Testing and verification, be it statistical or formal, is a founding tenet of all engineering disciplines, including software. Regardless, NIST reported in 2002 that software errors cost the US economy nearly US$60 billion [3]. That isn't because of failures in testing or verification *alone*, but is because of a systemic error or disconnect (then as it is now) in

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0002095

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
15 JAN 2015	**N/A**	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Engineering High Assurance Distributed Cyber Physical Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Klein /Scott Hissam David S. Kyle Sagar Chaki Dionisio de Niz Jeffery P. Hansen Gabriel Moreno Mark	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**4**	

how software-intensive systems are engineered ("from beginning to 'failure'"). The failure is in the integration of all the sub-disciplines that need to be integrated at the time a system is conceived from inception to transition (in Rational Unified Process parlance) and into the system's execution in the real world. It is here, in the real world, where resiliency in the face of uncertainty needs to be handled adequately and safely—something that cannot be exhaustively tested *a priori*.

Since the 1980's, research and development in the fields of Computer-aided Software Engineering (CASE), Model-based engineering (MBE), Model-driven engineering (MDE), Model-centric software engineering (MCSE), and others have attempted to leverage and integrate techniques for requirements, environment specification, architecture definition, domain-specific languages, design patterns, code-generation, analysis, test-generation, and simulation and emulation to support system development [4]. Further, in [4], Schmidt recognized the challenges to MBE (generalized to all such approaches) to include synchronization between the models and source code, debugging at the model level, expression of the design intent, and quality of service properties and the certification of safety properties.

Wallnau's work on *predictability by construction* [5] made two notable contributions with respect to these challenges. The first was treating quality attributes (non-functional requirements) as first class design elements suitable for design time and runtime analysis (timing analysis and safety properties). The second contribution was a component language demonstrating the capacity to (a) produce and evaluate an analytic model of a system based on quality attributes; and (b) generate executable code for the system if its analytic model did not violate its quality attribute requirements [6][7]. The intent for this work was to narrow the gap between architecture design specifications, quality attribute-specific analytic models and the enforcement of those specifications, through automation, to the generated code.

Feiler and Gluch's work on the SAE Architecture Analysis & Design Language (AADL) is used to model both software and hardware for embedded, software-reliant systems [8]. It is intended to support MBE analysis practice to encompass software system design, integration, and assurance. Furthermore, the standard is extensible to additional analysis and specification techniques necessary for domain-specific application.

Rainbow [10], based on an external, feedback control approach called the Monitor-Analyze-Plan-Execute (MAPE) model [9], is an architecture-based self-adaptation framework that maintains a model of the architecture of the running system at runtime and uses that model to reason about the adaptations that should be made to the system to achieve desired quality attributes. Rainbow's contribution provides an MBE approach to monitor a target system and its environment, reflect observations into a system's architecture model, detect opportunities for improvement, select a course of action, and effect changes during runtime in a closed loop.

III. DART Software Engineering Approach

The approach to engineering a DART system is consistent with that generally exhibited in MBE, one that *sufficiently* combines the *process* of engineering with the *tools* for engineering. Sufficiently here is intended to denote that the rigor needed for engineering (w.r.t. the processes and tooling) will differ based on the needs of that which is to be delivered. For a DART system, those needs call for high assurance safety-critical properties that can be formally verified as well as application-critical properties that can be objectively quantified.

The DART software engineering approach is process agnostic, as it does not matter if the process is waterfall or iterative, team-based or Scrum-based, heavy-weight or light-weight when working through requirements, design, analysis, and test to deliver the system [11]. What matters is that the safety-critical properties of the system are not violated, and are assured by objective evidence. Thus, the DART approach subscribes to a double V model (*build the system*, the first V, and *build the assurance case*[2], the latter V) that focuses on the artifacts that are produced during the build process and the type of analyses conducted on those artifacts **Error! Reference source not found.**.

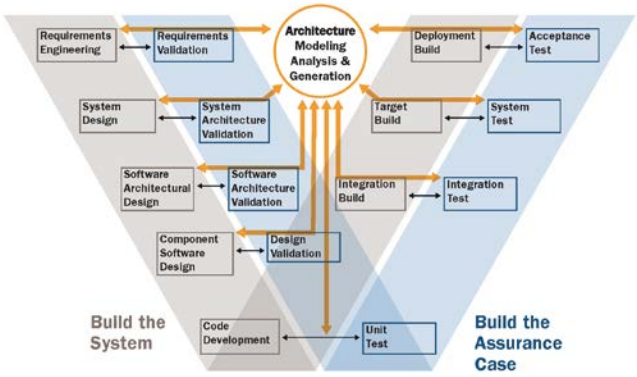

Figure 1: Double V Model for Validation & Verification

From a tooling perspective, DART seeks an Integrated Development Environment (IDE) that supports the specification of requirements, architecture and code design elements, tests (both unit and integration), along with code generation, compilation, deployment and debugging. Further, it is necessary to be able to trace such specifications backwards and forward through all the transformations supported by the IDE. For example, the Android Software Development Kit (SDK), an Eclipse-based IDE, is very strong at the tail end of the MBE tool-chain (i.e., from (GUI) design and code specification, unit and integration test specification, code compilation, debug, and deployment) but lacks support for the specification of requirements and higher-level architectural specifications. On the other hand, the Open Source AADL Tool Environment (OSATE), also an Eclipse-based IDE, is rooted in safety-conscience engineering supporting the double V model. As such, OSATE's plug-in architecture and extensions (e.g., the SAE AADL Error Model Annex) is structured for inclusion of domain-specific and quality attribute specific analysis tools. For example, ALISA, a plug-in for OSATE currently under development, supports the specification of goals, requirements, and claims, concepts of obstacles, hazards, vulnerabilities,

[2] An assurance case is a method to systematically manage objective evidence and arguments (e.g., reviews, analysis, and testing) that take into consideration the context and assumptions of the system being delivered [11].

challenges, and defeaters, and concepts of static analysis, verification activity, evidence, and counter evidence.

In the context of the double V model, AADL, OSATE and Rainbow, the DART software engineering approach is integrating:

- mixed-criticality analysis to verify the assymetric timing protection and schedulability of threads with different criticality that share resources (e.g., cpu(s)) in a single node [13];
- domain-specific language and safety specification notation for distributed applications comprising multiple nodes [14];
- model-driven verifying compilation system which generates C++ code if the safety properties specified for the application are verified successfully by a software model checker [14];
- latency-aware self-adaptation mechanism as a means for assuring resiliency when dealing with planned mode changes and unexpected events from the physical environment during runtime [15]; and
- statistical model checker for computing the bounded probability that best-effort properties of the system are within the application's requirements despite the stochastic behavior of the environment [16].

A complete demonstration of an integrated approach to engineering a DART system is staged in two phases. The first phase is to use the analysis techniques listed above and incrementally improve upon them to address their respective limitations when applied to a DART system. The second phase will use the lessons from those improvements to drive requirements and improvements (or extension) to AADL (e.g., ALISA) so that the safety-critical properties verified during the first phase can be properly encoded and traced through OSATE to engineer a complete end-to-end DART system.

IV. DART Model Problem

The model problem, which serves as the basis for both phases, involves collaborating swarms of autonomous agents (i.e., UAVs) that require both guaranteed, and best-effort requirements. Figure 2 depicts two fleets of agent-based swarms, each having independent objectives but sharing the same goal (e.g., search and rescue). The focus for the phase 1 model problem considers only one swarm and its objectives. The swarm is made up of a number of agents. Agents within the swarm must collaborate to maintain separation so as not to collide with one another (a safety-property) while maintaining a formation so as to best protect a "leader" (a best-effort property) during the time it takes to reach objectives along the swarm's route. We consider the distributed algorithm to maintain separation the highly critical property to be guaranteed and the protection property to be the low(er) critical property we also want to satisfy. In all cases, the real-time high-critical deadlines cannot be missed.

Figure 3 shows the initial tool chain for phase 1. The tools and techniques are founded on those discussed in the previous section, that is [13] through [16]. System level specifications are encoded in our domain-specific language (DSL) from [14]. Specification of application level requirements, and the environment come from subject matter experts. Initially, that to is manually crafted into our DSL so that the necessary verification steps can be performed. In Phase 2, the idea is to encode and formalize that knowledge in AADL.

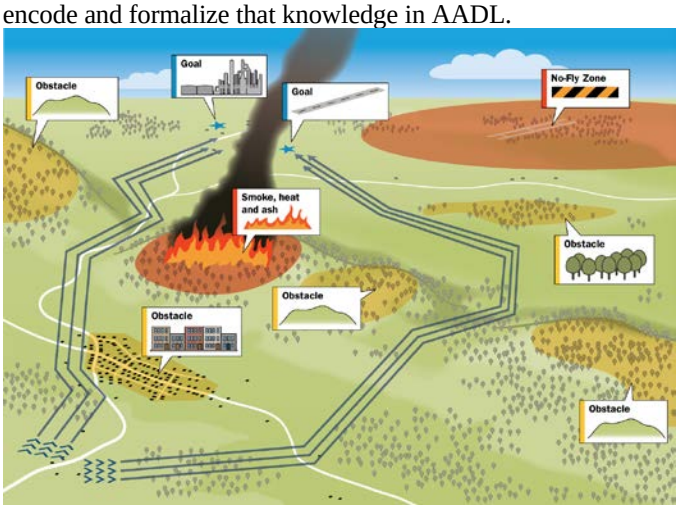

Figure 2: Context for DART Model Problem

Continuing in Figure 3, each verification tool takes its respective inputs from the specifications to perform the analysis (i.e., timing, functional, and probabilistic). If verification fails, a trace back to the specification(s) that formed the basis for a failed check is identified. Here, verification takes these three forms:

1. mixed-criticality temporal protection mechanisms between runtime threads hold; passing these checks means that real-time deadlines for high-critical tasks are guaranteed to be met,
2. guaranteed property: physical separation among multiple agents; passing these checks mean that the invariants for the distributed collision avoidance algorithm used by the swarm hold given the environment, and
3. best-effort property: agents provide adequate physical protection to the leader in a particular environment; passing these checks mean that over a given mission time, the probability that the protection is maintained is above a specified threshold.

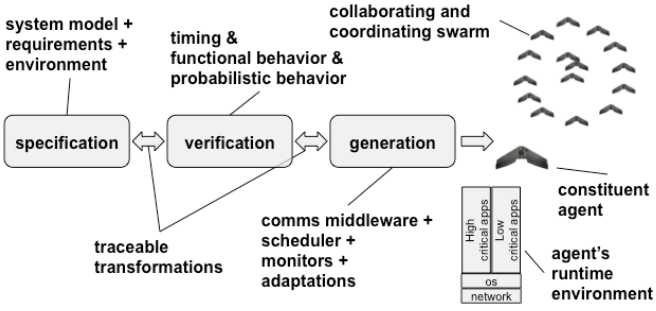

Figure 3: Abstract Engineering Tool Chain for DART (Phase 1)

Code is generated for the target hardware platform after passing the verification checks (see Figure 4). It includes all the functional code (allocated to one or more threads as identified

from the specification) and interfaces to the underlying operating system scheduler and networking services. Additionally, monitoring code supporting each of the requirements is generated:

- guaranteed requirement: when the `ASSERT()` based on the `require()` property could not be verified (due to scalability of the model checker). However, if the property is model checked successfully, then no monitoring code is generated.
- best-effort requirement: when the variables used to evaluate the `expect()` property specification that are also used by the self-adaptation mechanism are passed to the adaptation manager's monitoring interface.

For properties that require analysis across variables spanning more than one node (e.g., the `require (FORALL_NODE_PAIRS)` node specification in Figure 4), it may be impractical to share those variables across those nodes at runtime—for now no monitoring code for the target platform is generated in those cases. In the case that a property is deemed intractable and node-spanning, this generates a warning that would need to be addressed by a human, requiring changes in requirements or design.

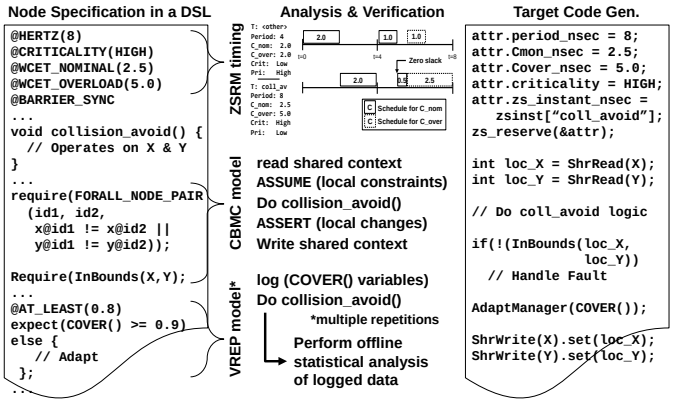

Figure 4: Code Generation based on Specifications and Analysis

The initial self-adaptation mechanism for the model problem deals mainly with the decision to change the formation of the swarm. Different formations provide tradeoffs between different qualities (e.g., protection vs. speed), which are desired for different phases of the mission. The adaptation mechanism must deal not only with planned mission events, but also with uncertain environment conditions (e.g., an unplanned forest fire that is on the current route must be avoided).

V. DART Research Agenda or Future Work

Both phases are intended to build upon the properties that can be verified for the individual agent as well the composed swarm of agents. In Phase 2 our work will be extended to properties that can be verified among the fleet of swarms. Further, we expect to:

- encode our DSL (i.e., `expect`, `require`) as AADL requirement specifications which are then mapped to generated specifications for both analytic models and code and to support debugging and back tracing.
- account for IO and Interrupts in timing analysis.
- introduce machine leaning for the adaption manager.
- support asynchronous mutli-agent coordination in guaranteed behavior as it applies to unbounded checks.
- reduce the total number of samples needed for a given level of precision for statistical model checking.

Finally, more research will be necessary to extrapolate from the lessons learned and discoveries made from the research presented here and from others in the community to scale software engineering to more numerous, interconnected critical systems.

References

[1] Mills, H. D. "The Management of Software Engineering Part I: Principles of Software Engineering." IBM Syst. J. 38, 2-3, pp.289-295, June 1999.

[2] Xie, T, "Software Engineering Conferences", web page http://web.engr.illinois.edu/~taoxie/seconferences.htm, January, 2015.

[3] Tassey, G., "The Economic Impacts of Inadequate Infrastructure for Software Testing ", Technical Report NIST 2002-10, National Institute of Standards and Technology, May 2002. http://www.nist.gov/director/planning/upload/report02-3.pdf

[4] Schmidt, D.C. "Guest Editor's Introduction: Model-Driven Engineering," IEEE Computer 39 (2), pp. 25-31, February, 2006.

[5] Wallnau, Kurt C. "Predictability by Construction: Working the Architecture/Program Seam," Mälardalen University Press Dissertations, No. 85, September, 2010.

[6] Moreno, G. A., and Hansen, J., "Overview of the Lambda-star Performance Reasoning Frameworks." CMU/SEI-2008-TR-020, Software Engineering Institute, Carnegie Mellon University, Feb. 2008.

[7] Chaki, S., Ivers, J., Sharygina, N., and Wallnau, K. "The Comfort Reasoning Framework". 17th International Conference on Computer Aided Verification, Springer, LNCS, vol. 3576, July 2005.

[8] Feiler, P. H. and Gluch, D. P., "Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language," Addison-Wesley Professional, 2013, ISBN: 9780321888945 2012.

[9] Kephart, Jeffrey O., and David M. Chess. "The Vision of Autonomic Computing." Computer 36.1 (2003): pp. 41-50, January 2003.

[10] Garlan, D.; Schmerl, B.; & Cheng, S-W. "Software Architecture-based Self-adaptation." In Autonomic computing and networking, pp. 31-55. Springer, 2009.

[11] P. Bourque and R.E. Fairley, eds., "Guide to the Software Engineering Body of Knowledge", Version 3.0, IEEE Computer Society, 2014; http://www.swebok.org

[12] Feiler, P., Goodenough, J., Gurfinkel, A., Weinstock, C., Wrage, L., "Reliability Validation and Improvement Framework", CMU/SEI-2012-SR-013, Software Engineering Institute, Carnegie Mellon University, December 2010.

[13] de Niz, D.; Lakshmanan, K.; Rajkumar, R., "On the Scheduling of Mixed-Criticality Real-Time Task Sets," Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE, vol., no., pp.291-300, December. 2009.

[14] Chaki, S., Edmondson, J., "Model-Driven Verifying Compilation of Synchronous Distributed Applications," Model-Driven Engineering Languages and Systems (MODELS), Springer, LNCS, vol. 8767, pp. 201-217, October 2014.

[15] Cámara, J., Moreno, G. A., and Garlan, D., "Stochastic Game Analysis and Latency Awareness for Proactive Self-adaptation," 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014). ACM, New York, NY, USA, pp. 155-164. May 2014.

[16] Hansen, J.P., Wrage, L., Chaki, S., de Niz, D., and Klein, M., "Semantic Importance Sampling for Statistical Model Checking," *to appear in* Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Springer, LNCS, April 2015.